CS575 Parallel Computing Project#7B
Autocorrelation using CPU OpenMP, CPU SIMD, and GPU OpenCL

Shangjia Dong

[1]School of Civil and Construction Engineering, Oregon State University

dongs@oregonstate.edu

1.

Question: What machines you ran this on

Response: I ran OpenMP and SIMD on Flip:

- Model name: Intel(R) Xeon(R) CPU X5650 2.67GHz
- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- CPU family: 6
- Model: 44
- L1d cache: 32K
- L1i cache: 32K
- L2 cache: 256K
- L3 cache: 12288K

I ran OpenCL application on Rabbit:

- 2 E5-2630 Xeon Processors
- ArchitectureArchitecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- CPU(s): 32
- Thread(s) per core: 2
- Core(s) per socket: 8
- Socket(s): 2
- 64 GB of memory
- 2 TB of disk

- GPU: NVIDIA Titan Black
- 2880 CUDA Cores
- 6GB of memory

2.

Question: Show the Sums[1] ... Sums[512] vs. shift scatterplot.

Response: See Figure 1

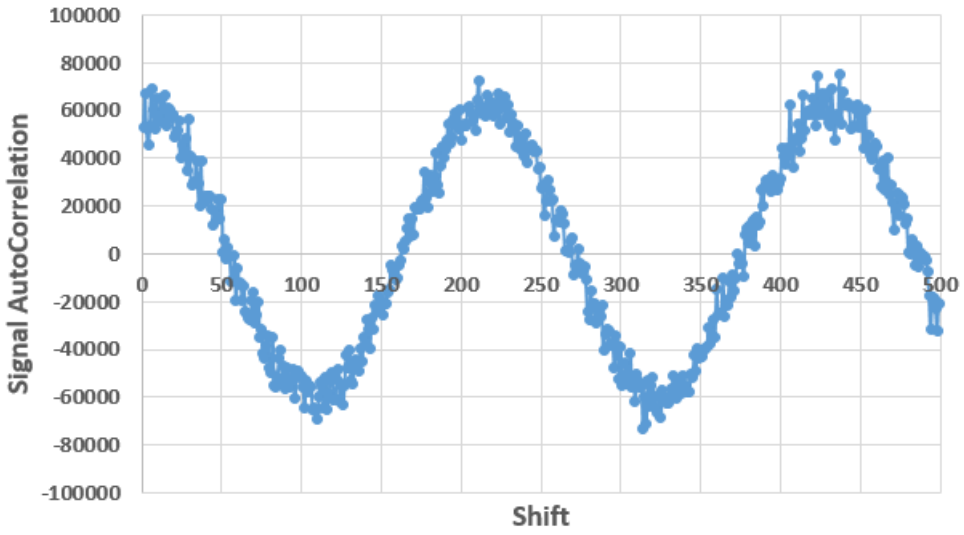

Figure 1. Signal AutoCorrelation vs. Shifts

3.

Question: State what the hidden sine-wave period is, i.e., at what multiples of shift are you seeing maxima in the graph?

Response: As seen clearly in the figure 1, there is a sine-wave pattern in the autocorrelation plot. The autocorrelation hits the minimum at 110, and arrives at mamima around 220. Thus the cycle length is around shift ≈ 220.

4.

Question: What patterns are you seeing in the performance bar chart? Which of the four tests runs fastest, next fastest, etc.? By a little, or by a lot?

Table 1. Performance of the investigated applications

OpenMP	SIMD - SEE Width 4	OpenCL
1-thread: 261.36	1768.88	Local Size 32: 42093.478
4-thread: 795.69		Local Size 64: 53335.310
8-thread: 1583.06		Local Size 128: 52210.919
16-thread: 2130.04	Unit (MegaMultsPerSecond)	

Response: OpenCL runs the fastest, second is the SIMD, and the slowest is the openMP. Within the openMP application, 32-threads runs faster than 1-thread case. Among the openMP 1-thread, openMP 32-thread, and SIMD, the performance varies, and the difference is relative big. But when we considering the openCL, the difference is huge.

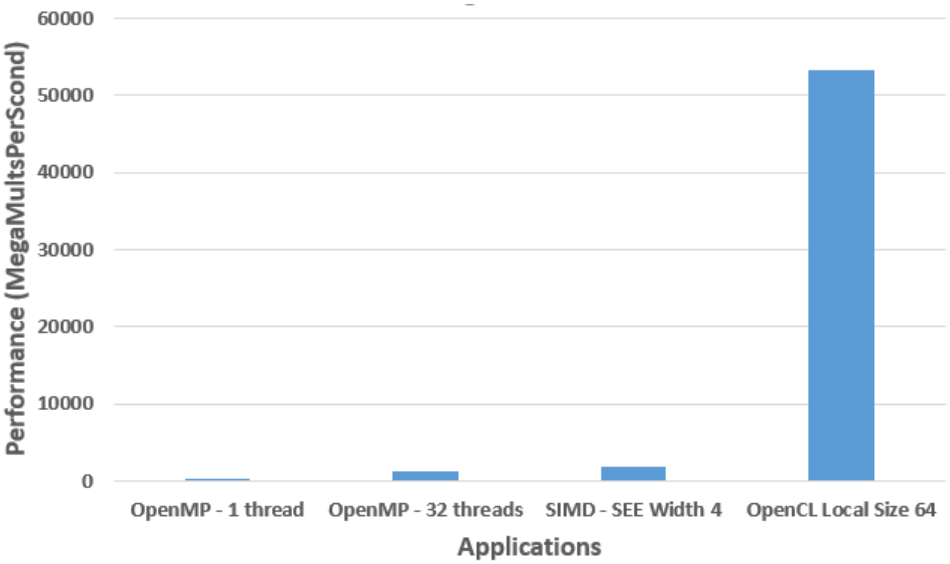

Figure 2. Performance comparison of four applications

5.

Question: Why do you think the performances work this way?

Response:

Nowdays, computers normally have multiple cores, OpenMP takes advantage of this feature. When we only using 1-thread, as in first case, some of the core is idle, we are not fully utilizing the resources. When we use #pragama omp, the tasks can be distributed to different threads, and thus all the cores are utilized and the perforamance increases.

While each of the core has its own SIMD registers, this is when SIMD plays its advantage. In this specific problems, we are just doing simple multiplication and summation, which can be vectorized. Instead of retrived one elements at a time and do the multiplication and add, SIMD gives a single instruction says ”retrive # size of the elements’, which take significantly less time. In addition, in this case, SIMD load 4 data points at once, and the SimdMulSum will compute them at the same time. Also, The parameter ”sum” is stored in the memory, this makes it really fast to fetch. Plus, the assembly code is saved on the register, this leads to the faster performance. That’s why its performance is better than openMP.

OpenCL is at its best on compute devises with large amounts of data parallelism. The problem is broke into lots of pieces, and each piece gets farmed out to threads on the GPU. Different from CPU, GPU has a huge number of CUDA cores, and they can reach 192 cores per streaming multiprocessors (compute unit). It is like 192-way SIMD, which is humongous. Therefore, when we parallelize a problem in openCL, the performance will get a huge bump. Of course, it depends on the problem. In this case, we are only doing multiplication and sum, which is a very parallelable problem.